DEVELOPMENT OF A WEB-BASED TUTORIAL FOR VISUAL BUILDER --- THE GUI DESIGNER IN IBM VISUALAGE COBOL

By

JIAZHENG LI

Bachelor of Engineering
Tsinghua University
Beijing, China
1984

Master of Science
Nanjing University
Nanjing, China
1987

Doctor of Philosophy
Oklahoma State University
Stillwater, Oklahoma
1997

Submitted to the Faculty of the
Graduate College of the
Oklahoma State University
in partial fulfillment of
the requirements for
the Degree of
MASTER OF SCIENCE
May, 1999

DEVELOPMENT OF A WEB-BASED TUTORIAL FOR

VISUAL BUILDER --- THE GUI DESIGNER

IN IBM VISUALAGE COBOL

Thesis Approved:

Jacques E. LaFrance

Thesis Advisor

J Chandler

Wayne B. Powell

Dean of the Graduate College

ACKNOWLEDGMENTS

I would like to extend my sincere gratitude to Dr. Jacques LaFrance, my principle advisor, for his guidance, ideas, and expertise in helping me to develop and complete this study. I wish to tell him how much his efforts have meant to me, since he not only provided me with this research opportunity and guided me toward a Master's degree in a wonderful discipline, but also directed me developing a career. Special thanks go to Dr. John Chandler and Dr. K. M. George for their patience, support, and services as members of my advisory committee.

Sincere thanks are also extended to all my professors at Oklahoma State University from whom I have gained so much, especially those in Environmental Engineering where I used to study and work for over four years.

Many of my friends and/or fellow students, including Mei Wang, Haobo, Haihui, Shaokai Wen, Rui Zhang, Alan, Sungwook, Gary, and Paul, deserve special thanks and acknowledgments. Without their help, encouragement, and friendship, my study on this thesis could have been much more difficult with much less fun.

This thesis is dedicated to my mother and my late father. It is their love, support, teaching, encouragement, and expectation that have kept me motivated and escorted me in all my endeavors through the years.

TABLE OF CONTENTS

LIST OF TABLES AND FIGURES

CHAPTER ONE

INTRODUCTION

COBOL and Visual Builder in VisualAge COBOL

COBOL was one of the earliest high-level computer languages. It was first developed in 1959 by a group of professionals. Since then it has undergone several modifications and improvements and been used effectively, especially for business-oriented applications. For the following three decades after COBOL was first developed, it was the worldwide leader as the most commonly used computer language [Nickerson, 1987; Chapin, 1997].

Most earlier computer manufacturers implemented a slightly different form or version of the COBOL language for use with their own computers. In an attempt to overcome this problem of incompatibility between versions of COBOL, the American National Standards Institute (ANSI) developed a standard form of the language in 1968. A revision was released in 1974. In 1985 ANSI published another revised version (ANSI COBOL 85). COBOL's desirable ability to manipulate data makes it a good language in dealing with vast amounts of data that businesses process. It is ideally suited for business application development, maintenance, and production support [Longhurst, 1989]. Even though the front end of software systems, for the past decade, has been largely converted

to PC-based graphical user interfaces (GUIs) developed in various computer languages, the ultimate destination of the data is often an older system written in COBOL. It is estimated that there are approximately seventy billion lines of COBOL code in use and, as of 1995, COBOL was still used in over sixty five percent of newly developed applications [IBM-ITSO, 1996]. Since the industry has made a huge investment in the installed base of COBOL code and the people who code and maintain COBOL systems, COBOL language will remain an important tool of choice for most business applications [Levey, 1996]. Another issue that renews people's interest in COBOL is the "millennium bug", or Y2K problem. Most of the 1960s' and early 70s' business-oriented systems were written in COBOL; and it was a standard practice, until recent years, to use the six-digit date (two for years) in order to save storage space. Making these systems Y2K-compliant largely relies on finding the bug in the code, which means significant amount of work in COBOL re-programming and maintenance [Herman, 1997].

In the past decade, object-oriented programming and GUI design have become very popular and important in software development. Object orientation has emerged as an effective approach to developing complex software by decomposing a large problem into smaller subgroups and building/reusing individual components. Software designers have enriched the presentation and facilitated the use of software on personal computers and workstations by providing users with GUIs [Carrel-Billiard et al., 1996]. These new trends in software engineering could be a big challenge to a relatively old language such as COBOL. However, the good news is that COBOL has been modified to accommodate both object orientation and GUI presentation [Chapin, 1997; Carrel-Billiard, et al, 1996]. The latest ANSI standard COBOL is featured with object-oriented capability while

leaving almost all of COBOL 85 intact and still available for use [Chapin, 1997; IBM-ITSO, 1996]. In addition, VisualAge COBOL, developed by IBM, makes it possible to develop GUI enriched COBOL applications in an object-oriented way. This is a member of IBM's VisualAge family, and a new COBOL development environment on OS/2, Windows 95 and Windows NT. All of the IBM COBOL family of solutions support the high subset of ANSI COBOL 85 functions, so the applications can be ported across supported platforms including mainframes and personal computers with OS/2, Windows 95, or Windows NT [IBM, 1997a].

VisualAge COBOL provides software developers with a visual software development environment, analogous to Microsoft Visual C++ environment used in C/C++ programming. The VisualAge COBOL environment includes a visual builder (or GUI designer), several editors, and a debugger, among other features, which enable programmers to create GUI applications. VisualAge COBOL also supports object-oriented extensions, allowing programmers to develop discrete software objects and to share System Object Model (SOM) -enabled objects created by other languages, such as C++ [IBM, 1997b].

Because of the widespread and continuing use of COBOL, developing a curriculum that offers lectures in COBOL programming, especially programming in a visual development environment such as VisualAge COBOL, is of particular significance and interest. This study is to design and implement a Web-based tutorial for Visual Builder, as a part of a training curriculum for VisualAge COBOL. Visual Builder, as a GUI designer, is the major visual development tool in the IBM VisualAge COBOL (for Windows NT) software package. In fact, Visual Builder is largely language-independent;

it is also used as the visual development tool in IBM VisualAge C++ [IBM, 1998a]; and a very similar tool is used in IBM VisualAge for Java [Williams, 1998]. Visual Builder is considered a powerful programming tool since it not only helps build complete visual applications in COBOL but also enables the programmer to adopt the object-oriented technology immediately, the reason being that Visual Builder uses a "construction from parts" approach and that a part is simply a software object [IBM, 1998b].

Using Web-based tutorials to teach programming tools and techniques is a relatively new form of distance education. It takes advantage of the easy accessibility and the satisfactory effectiveness of distance learning and education, as discussed in Chapter Two, and involves designing and implementing online courseware applications. Authoring a Web-base tutorial, just like developing other Web pages, can be done in different ways, including writing Web pages in HTML directly and embedding various controls (e.g. Java applets or ActiveX controls written in C++) in the pages, programming in multimedia presentation-oriented languages [Bouthillier, 1998], or using Web publishing/authoring tools. The ToolBook II Instructor software package, developed by Asymetrix Corporation, is an authoring tool mostly used in developing Internet courseware. Because of its support for multimedia, easy-to-use tools, and capability of creating HTML documents, ToolBook II Instructor is a comprehensive software for authoring, distributing, and managing online multimedia courseware [Asymetrix, 1996]. It is based on a book paradigm so that the curriculum developed under this environment is divided into chapters and pages. It fits the courseware nature of the Visual Builder tutorial properly and therefore is used as the development tool in implementing the tutorial in this study.

Objectives of This Study

Since VisualAge COBOL is a new and useful development environment for COBOL programmers and no Web-based courses or tutorials have been found to cover this topic systematically, developing a Web-based tutorial for VisualAge COBOL is of substantial significance [Wang, 1998]. Based on the above discussion, the objective of this study is to design and implement a Web-based tutorial for Visual Builder in VisualAge COBOL. The tutorial teaches basic concepts and use of the editors and tools in Visual Builder, and discusses development of object-oriented, GUI-enriched COBOL applications using Visual Builder. In order to provide a solid knowledge basis that supports the development of the Visual Builder tutorial, relevant research, largely reflected in the literature review in Chapter Two, has been conducted in the following three areas: 1) investigating visual interface design principles, object-oriented programming approach, and their realization through the Visual Builder functionality; 2) examining the feasibility and effectiveness of offering Web-based courses in Visual Builder; and 3) experimenting ToolBook II Instructor as a new tool in courseware authoring.

CHAPTER TWO

LITERATURE REVIEW

Graphical User Interface and Visual Builder

For years, the end users of computer software have expressed their frustration with complicated interface procedures and incomprehensible screens. In recent years, finally, greatly improved technology has eliminated many of the barriers to good interface design and unleashed a variety of new display and interaction techniques wrapped into a package called the Graphical User Interface (GUI). In brief, a user *interface* is a collection of techniques and mechanisms to interact with the person operating the computer. In a *graphical* interface, the primary interaction mechanism is a pointing device which is the electronic equivalent to the human hand. What the user interacts with is a collection of elements referred to as *objects*. People perform operations, called actions, on objects. [Galitz, 1997].

The success and popularity of graphical systems have been attributed to a host of factors. First of all, visual symbols are recognized faster and more accurately than text [Ells and Dewar, 1979]. The graphical attributes of icons such as shape and color are very useful for quickly classifying objects, elements, or text by some common property. Also, a visual, graphical representation has been found to aid faster learning as well as

easier use and problem solving [Carroll et al., 1980]. Other advantages of a graphical interface include easier teaching, understanding, and remembering, increased feeling of natural interaction and control by humans, immediate visible feedback, more attractive software, lower typing requirements and use anxiety, among others. On the other hand, some studies and findings have challenged that graphical representation and interaction may not necessarily always be better than pure textual displays. People have indicated some disadvantages of GUI representations, which include greater design and coding complexity, lack of experimentally derived design guidelines, inconsistencies in techniques and terminology, window manipulation time requirements, inefficiency for expert users, and hardware limitations [Galitz, 1997].

The design goals and principles in overcoming the above disadvantages and creating a desirable GUI have been investigated and described by many researchers. It is very essential to understand how people interact with computers and what the needs of the end user are. Humans are complex organisms with a variety of attributes that have an important influence on screen design. These factors include perception/awareness for visual objects, memory, visual acuity, learning ability, etc. [Tullis, 1983]. A poorly designed computer user interface may make people feel that it is talking in a strange and confusing language or logic. Some terms, words and conventions may seem to be very clear and natural to programmers and designers but become completely alien to users in a different environment or context. Also, a task analysis is important in interface design in order to understand the current user activities and requirements. Task analyses may be accomplished through direct observation, interviews, questionnaires, or obtaining references and measurements of actual system usage. What are people looking for in the

design of screens? One organization asked a group of screen users and obtained the following results [Galitz, 1997]:

- An orderly, clean, clutter-free, and aesthetically pleasant appearance.
- An obvious indication of what is being shown and what should be done with it.
- Information to be found in an expected location.
- A clear indication of the relationships among entities.
- Plain, simple human language.
- A simple way to find out what is in a system and how to get it out.

GUI design is also affected by the physical characteristics of the hardware. The design must be compatible with hardware capabilities such as system power, screen size, screen resolution, and displayable colors. Design must also be compatible with the system platform and development/implementation tools being used. Myers and Rosson [Myers and Rosson, 1992] report that about 50 percent of software code is now devoted to user interface design. Available tools include various tool kits, interface builders, and user interface management systems.

Software packages and tools called GUI designers/builders, as mentioned above, have been developed to help application developers design and create GUIs more efficiently. Mostly, GUI builders let programmers create the graphical interface of an application by dragging and dropping GUI objects, such as check boxes, onto a window object. However, traditional GUI builders do not provide a visual way of generating the

behavior of the application, such as the piece of code executed when a push button is clicked. On the other hand, Visual Builder in IBM VisualAge is a GUI builder that enables programmers to create a complete application visually by creating and reusing parts (or objects), connecting them, and generating the code for the application [Carrel-Billiard et al., 1996].

Visual Builder has adopted a "construction from parts" approach to software development (this is further discussed in the next section). In this concept, any application made from parts is a part itself. The Composition Editor in Visual Builder is a workspace for the application developer to layout visual parts and design the visual interface for the application. GUI applications are often event-driven. In order to achieve this, the Composition Editor allows the designer to make various connections between parts. For example, a connection can be made between a push button and a list box. When the *button-press* event occurs, it may trigger a certain action (e.g. search a linked list to find certain data) and put the results of the action in the list box. The Class Editor in Visual Builder helps the designer edit files and resources associated with a class, which is the logical (or abstract) representation of a part. The Part Interface Editor in Visual Builder allows the designer to define and modify a part's interface. A part interface consists of three types of features --- attributes, events, and actions --- and defines the nature and behavior of the part. These features make connections between parts possible and workable [IBM, 1998b]. Visual Builder is also capable of generating code automatically using the IBM Open Class Library. More or less similar to the Microsoft Foundation Classes, the IBM Open Class Library provides a comprehensive range of generalized and reusable classes from which programmers can create code for software

objects and manipulate them [Carrel-Billiard et al., 1996]. In many relatively simple cases, the original code generated automatically by Visual Builder is so complete that it is able to support a fully functional application. In other situations, the programmer may need to edit/ customize the machine-generated code and declare/implement more functions. For example, suppose we are developing an application that has a push button and a static text field on its GUI. When the user clicks on the button, the text field is to display the current date. To achieve this, we can declare a function *getDate* as a member method of the GUI window class using the editors in Visual Builder, make connections between the push button and the text field, and then have Visual Builder generate the skeleton COBOL source code automatically. Such a piece of machine-generated code is displayed in the code editor window shown in Figure 2-1. Based on the skeleton method, the programmer can add more code (e.g. code that fetches the input data as the current date) to implement the desired functionality.

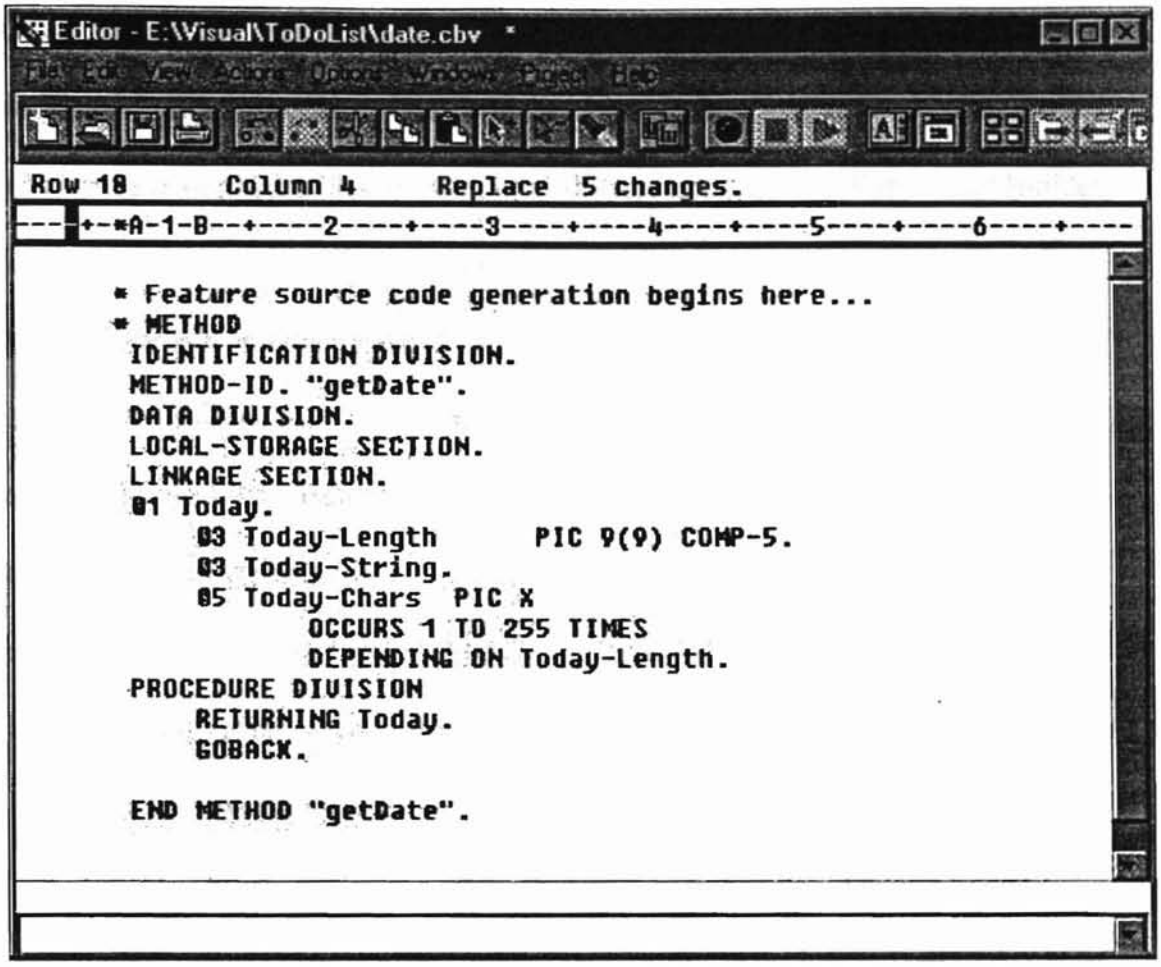

Figure 2-1. COBOL source code generated by Visual Builder

Parts and Object-oriented Approach

As mentioned earlier, Visual Builder uses parts to build applications. These parts are reusable components that can be tailored, organized, and connected to fit the application needs. A primitive part can be a window or an entry field --- it is also called a "control" [Carrel-Billiard et al., 1996]. These parts or controls, when visible, are the objects on the user interface. Visual Builder provides the designer with a visualized way

of manipulating and organizing these objects to create visual user interfaces. This means that Visual Builder plays a crucial role in the object design (as discussed below), which consists of a major part of design work of a software application. With Visual Builder, parts can be divided into two categories, visual and nonvisual parts. Nonvisual parts often involve data processing and do most of the behind-the-scene work to support the functionality of visual parts. The visual parts are what we see as GUI controls on the user interface. Using Visual Builder, the designer can create visual parts in a bottom-up approach to achieve part reusability. That is, we first build the primitive or elementary visual parts (such as entry fields, list boxes, and push buttons, often supported by nonvisual parts). Then we can aggregate these parts to build more complex views (or composite parts), which represent the final assembly of the end-user interface.

Building applications from parts represents an object-oriented design and programming approach because, as we have seen so far, a part is just a software object. Object-oriented approach is a programming technique based on the concepts of data abstraction and inheritance. These concepts have been established mostly because of a goal that software engineers have been pursuing for a long time --- software reusability. However, this goal was seldom achieved until recent years. A major reason for this is the tight interconnectedness of most software constructed in a conventional manner. In conventional programming case-specific information is tightly bound with the more general code. On the other hand, object-oriented techniques provide a mechanism for cleanly separating the essential information (e.g. insertion and retrieval of data records) from the inconsequential information (e.g. the format for particular records). By using object-oriented techniques, just like building views from individual parts in Visual

Builder, software developers can construct large, complex applications from relatively small and independent, often reusable software components.

Alan Kay, considered by some to be the father of object-oriented programming (OOP) [Budd, 1997], identified the following characteristics as fundamental to OOP [Kay, 1993]:

1. Everything is an *object*.
2. Computation is performed by objects communicating with each other, requesting that other objects perform actions. Objects communicate by sending and receiving messages. A message is a request for an action.
3. Each object has its own memory, which consists of other objects.
4. Every object is an instance of a *class*. A class simply represents a grouping of similar objects, such as integers or lists.
5. The class is the repository for properties and behaviors associated with an object. That is, all objects that are instances of the same class can perform the same action.
6. Classes are organized into a singly rooted tree structure, called *inheritance* hierarchy. Memory and behavior associated with instances of a class are automatically available to any class associated with a descendant in this tree structure.
7. The implementation details of a class's working mechanisms (functions) can be hidden from other classes. This is the concept of *encapsulation*.
8. Different objects in a class hierarchy can be treated (e.g. allowed to make function calls) in the same manner; yet different actions can be performed to

respond to the different task requirements for individual objects. This is a very useful and important characteristic in object-oriented programming --- *polymorphism*.

Most of the effort to date in the object-oriented community has been focused on programming language issues. However, the object-oriented approach is a conceptual process independent of a programming language until the final implementation stages. It is fundamentally a new way of thinking [Rumbauch et al., 1991]. One of the greatest benefits of the object-oriented approach comes from helping designers, developers, and customers express abstract concepts clearly and communicate them to each other. Usually, the methodology of object-oriented development consists of building a model of an application domain and then adding implementation details to it. This is called the Object Modeling Technique (OMT), which includes four stages: problem analysis, system design, object design, and implementation.

Inheritance, encapsulation, and polymorphism are three vital and very useful features in the object-oriented design and programming approach. Inheritance allows the designer to declare a *bass class* containing the common properties and operations for certain categories of objects. Then one or more *subclasses* (or child classes) can be declared to contain special properties and operations for each kind of objects and, at the same time, to inherit the common features from the bass class. Figure 2-2 uses an example for vehicles to illustrate the concept of inheritance. Recognizing the commonality in the data objects in a problem is an important key to object-oriented design [Adams et al, 1995]. In this way, the redundancy in design and coding is largely

reduced; and the bass class, due to its commonality, has a good chance of becoming a reusable software component. A well-designed class with reasonable commonality and careful documentation can easily be reused in a later version or even in other completely separate programs. Programming with reusable components can also greatly simplify debugging. It is not uncommon that the greatest amount of time spent in a programming project is associated with finding the location and nature of bugs [Cannon, 1997]. Furthermore, the multiple inheritance technique allows a class to inherit directly from more than one bass class [Rogerson, 1997]. A subclass can be made very powerful and flexible by using multiple inheritance.

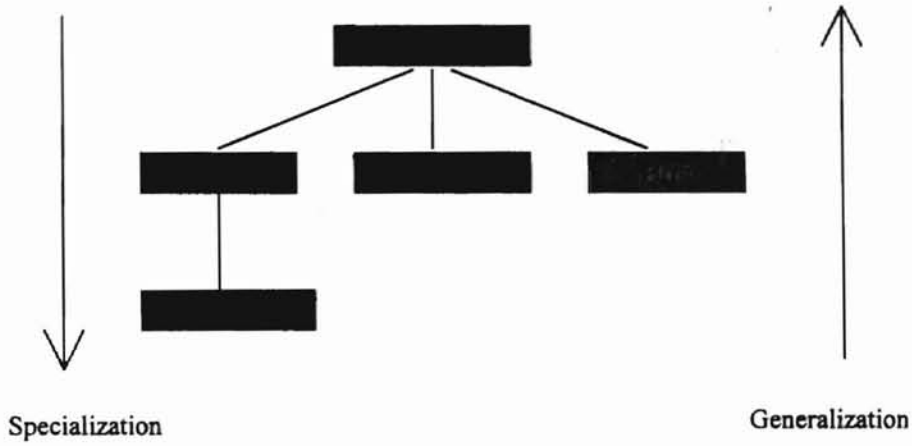

Figure 2-2. Illustration of inheritance

Encapsulation, also known as information hiding, is another major feature and benefit of object-oriented techniques. This makes it possible to store and hide various types of data and operations within class objects and only expose certain types of operations (declared as public member functions or interface methods) to the outside

world. Therefore, other class objects, components, or programs are forced to interact and communicate with a class object through its public member functions. The declaration of the set of public member functions can thus be thought of as an interface between this class and others. As long as this interface remains stable, any program or software component that uses the class solely through the interface need not to be modified, even if the inside implementation details of the class are modified extensively. Callers of the interface functions do not care how the underlying algorithms are implemented, but they rely on the promised behavior of the functions. This means that the maintenance of such a system is made more efficient and simplified. An encapsulated object can be upgraded without breaking its existing users [Adams et al, 1995; Rogerson, 1997]. In conventional, non-object-oriented languages, use of data structures (like those in C) can also be considered encapsulation. But the ability to combine data structures and operations in a single entity (class) in object-oriented languages makes encapsulation cleaner and much more powerful [Rumbauch, 1991].

When a function in a class hierarchy is called, the function actually executed can be the original version of a certain class, or it can be an inherited and altered (i.e. overridden) version in a child class, depending on which class's object is performing the function. This is polymorphism at work. In other words, the same action executed on different objects provokes different reactions. In pure object-oriented languages, a function is always coupled to a class. At coding time, the exact class that is coupled to the function for future execution need not (and often cannot) be known. Then, during run time, the class object that actually executes the invoked function can be one of the many class objects in the class hierarchy [Carrel-Billiard, 1996]. Notice that when we say a

"function" here, we are actually referring to a particular function name. To implement polymorphism in object-oriented COBOL and other object-oriented languages, all of the classes in a class hierarchy may have a particular function with the same name while the internals of each class's function differ so as to accommodate the differences in the class's data and tasks [Chapin, 1997]. Polymorphism may sound strange, but it does work because the link between data and functions is subject to delayed binding, often to run time. Usually one of the several versions of a function is bound to one of the several different data objects to produce different but appropriate results [Jezequel, 1996; Chapin, 1997]. Polymorphism, working together with class inheritance, makes it easier to upgrade and enhance functionality of an application. For example, in future releases of an application, one or more new subclasses might have been derived from a parent class to provide the user with new features. In each of these new classes, there may be a function whose name and prototype are the same as those of a function in the parent class while performing a new behavior. The good thing is that the caller of the function (e.g. a client) does not have to know about the existence of the new class but gets the new feature automatically because the function name was not changed! That is, polymorphism shifts the burden of deciding what implementation to be used from the calling code to the called class hierarchy.

Distance Education

This study involves lecturing concepts of object-oriented development, techniques in GUI design, and the use of Visual Builder in VisualAge COBOL to achieve the design

goals. The lectures are authored and to be delivered as a Web-based tutorial. Learners in various places and circumstances can access the tutorial through the World Wide Web. This, in fact, is a form of distance education and learning, which have provided opportunities to those who are seeking advanced or professional education at home or on the job whose multiple responsibilities and physical circumstances prevent them from attending a traditional course [Bates, 1995]. *Distance education* is a term that is often used interchangeably with *distance learning*. However, distance learning might best be seen as what takes place as a result of distance education [Connick, 1999].

Distance education meets various educational needs of people with various background and makes lifelong learning more feasible and more attractive. Learners participating in distance education programs may learn independently at their own pace. They may choose what they need from a greater variety of subjects that are, in turn, from a greater variety of institutions, and learn at a convenient time and location [Porter, 1997]. Distance education first started in the early 20th century. Traditionally, distance education meant teaching and learning through correspondence in written and printed materials. These types of distance education programs are still available. While the concept of distance education is not a new one, the types, means, and technologies involved in distance education have been changed and innovated continuously. Radio was used to deliver courses at a distance in the first half of the 20th century; and in the 1950s, local educational television stations developed. Since the 1960s, interactive video technologies began to gain popularity [Connick, 1999]. In recent years, computer technologies and the Internet have changed distance education dramatically. Disks, CDs, e-mails, Web pages, and online interaction provide a new and interactive means of

overcoming time and distance barriers to reach learners. One of the great benefits of learning on the Web is the use of hypertext and hypermedia to link textual documents and/or multimedia information all over the world. A US Department of Education study based on data from 1994-95 reported that three-quarters of large higher education institutions and two-thirds of medium sized institutions were then offering courses and educational programs at a distance. These proportions have further grown since then. The International Council for Distance Education has estimated that more than 10 million students are currently taking degree courses at a distance in the world and the number of people using distance education methods for other areas and levels of study (such as vocational and technical education, professional training) must be comparable, if not greater [Van den Brand, 1993]. Some higher education institutions have designed their distance education courses in such a manner that these courses closely parallel the institution's on-campus undergraduate and master's programs [Heilman, 1999].

One of the issues in distance education is how effective and successful it can be. Research comparing distance education with traditional classroom instruction has indicated that teaching and studying at a distance can be as effective as traditional instruction [Moor and Thompson, 1990]. The distance education and learning context does put special pressures on learners. That is, they must be independent and self-disciplined. Research has found that those who succeeded as distance learners are highly motivated and active in learning, have good organizational and time management skills, and can adapt to new learning environments [Connick, 1999]. While the degree of effectiveness in computer and Internet-based education may vary in different learning subjects and areas, people would naturally think of using computer technologies (e.g.

computer communication, the Internet, multimedia courseware) in teaching and learning computer technologies (e.g. programming languages, multimedia design). Imagine that a person is learning the use of Visual Builder in COBOL programming through a Web-based tutorial. He/she could have at least two major programs running on the computer at the same time. One is a Web browser that accesses and presents the tutorial, the other is VisualAge COBOL. He/she might be reading the contents in the tutorial for a while, and then trying to do some exercises in Visual Builder following the instructions in the tutorial. With a few more software applications running, he/she might also want to connect to the IMB Web site to find more references about VisualAge COBOL, look into various GUI applications and Web sites to get a flavor of various styles in GUI design, send an e-mail to ask a teacher or a "virtual classmate" questions, or even have real-time chatting and discussion with another learner at a distance. All these can be done virtually at the same time on a single personal computer. Helping provide the leaner with such a learning environment is a goal in the development of the Visual Builder tutorial in this study. In such a way of learning, two major aspects of knowledge, *explanation* and *experience*, are closely combined together [Hodges and Sasnett, 1993]. With integrated graphic, textual, audio, and video capabilities, computers can link various technologies and resources together providing learners with an effective and efficient learning environment. Computer and Internet -based learning also gives learners immediate access to various Internet discussion groups with information being exchanged from around the world. As indicated by Cummins and Sayers [Cummins and Sayers, 1995], effective (and also low-cost) learning networks have been formed globally on the basis of modern technologies.

Web-base Courseware Authoring and ToolBook II Instructor

The World Wide Web (or the Web) is a system of Internet servers that support specially formatted documents, HTML files. These files support links to other documents, software applications, and multimedia resources [Musciano and Kennedy, 1997]. The features of the Web enable distance educators to create documents containing hypertext/hypermedia links so as to provide learners with large amount of relevant information and references.

As Web-based instruction becomes more and more common, research and development have been conducted on reasonably designing and efficiently creating attractive Web-based curricula [Brooks, 1997]. A result of these efforts is the emergence of multimedia courseware authoring systems. These systems are software that helps Web-based educators and publishers design, create, edit, and deliver curricular documents. Similar to desktop publishing, courseware authoring involves combining source documents, including text files, graphics, video clips, and sounds, in a functionally effective and aesthetically pleasant format that communicates and shares knowledge. Therefore, authoring software serves as a tool that helps the course author put documents and information together and create the online course (or tutorial, courseware, etc.) in an effective and relatively easy way.

An attractive feature of Web-based multimedia courseware is two-way, interactive communication. Actually this has been the norm of communication among humans for thousands of years: one person talks, another responds. However, today's television, radio, newspaper, and books pour information into billions of passively receiving people

every day [Beekman, 1997]. Interactive technology offers new hope for turning communication back to a process of two- (or multiple-) way participation. In an interactive Web-based learning environment, the learner is part of the show. Just imagine this picture: instead of watching the professor flip overhead transparencies, the learner controls a self-paced presentation complete with text descriptions, images, cross-referencing links, and question/answer interaction, and is able to communicate with others through e-mail and online discussions. Researches have indicated that interaction and visualization can greatly help the development of problem-solving abilities [Haykin, 1994].

Courseware authoring tools vary widely in capabilities and user interfaces. Probably the most widely used interface for authoring tools is the card-and-stack interface originally introduced with Apple's HyperCard. A document system or a courseware created under the card-and-stack metaphor is virtually a stack of cards. Each screen is a card (or a page) and can contain text, graphics, hyper links, navigation buttons, etc. This metaphor has been adopted by a number of authoring systems, including Oracle Media Objects, HyperStudio, and ToolBook.

Asymetrix ToolBook II Instructor is an authoring software that helps courseware authors construct and deliver multimedia Web-based learning environments. ToolBook II provides the user with a programming language, OpenScript, as well as many tools and menu commands for courseware development. These tools and commands give the courseware author an easy and efficient way of creating commonly used controls in courseware applications (such as text fields, push buttons, etc.) without writing code --- ToolBool II generates the code behind these controls automatically! These tools and

commands are called *visual programming tools* and becoming more and more popular and available in today's software engineering field. They allow programmers to create large portions of their software programs by drawing/pointing to images and dragging/dropping on-screen objects, eliminating much of the tedious coding work of traditional programming. Although they have not completely transformed programming into a visual design process, their successful use has suggested that such a transformation is possible [Beekman, 1997].

To implement more advanced and complex functionality in a courseware application, the author can use ToolBook II's scripting language, OpenScript, for extensive, object-oriented programming. Scripting languages, such as OpenScript, VBScript, and HyperTalk, are high-level macro languages often built into development environments and/or operating systems. They usually allow the user to create macros that automate repetitive tasks, have powerful statements, and are relatively easy to use. OpenScript is also an object-oriented language. It is supported by the ToolBook II environment in a way similar to Visual C++ supported by Microsoft Windows. Execution of OpenScript programs is event-driven; an event (such as button click) that occurs to an object is signaled to this and other objects through a messaging mechanism. The object that receives a message needs to handle it and take appropriate action. This is done by a message handler of the object, which is an OpenScript routine that responds to a particular message. For example, we can make an animation with an object that, when clicked on, moves along a line. The message handler for the *buttonclick* event can be written in TookBook II's script editor and include statements as those shown in Figure 2-3.

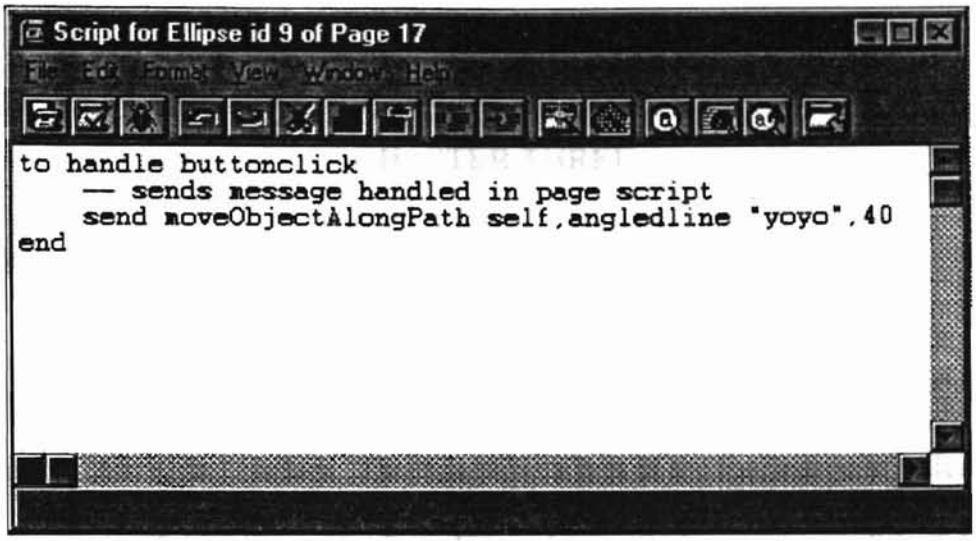

Figure 2-3. A message handler in OpenScript.

CHAPTER THREE

DESIGN AND IMPLEMENTATION

This chapter describes and discusses the design and implementation of the Web-based tutorial for Visual Builder. This tutorial is a part of the four-part curriculum for IBM VisualAge COBOL programming. The entire curriculum consists of the following portions: Part I: Object-oriented COBOL, introducing object-oriented COBOL language; Part II: Visual Modeling Techniques (VMT), presenting the construction-from-parts paradigm and a theoretical basis for object-oriented visual COBOL programming; Part III: Visual Builder, describing the features, functionality, and use of Visual Builder, a GUI designer in VisualAge COBOL; Part IV: VisaulAge COBOL, introducing the entire development environment and its tools. Parts I, II and IV have been completed or under development by other developers [Wang, 1998]. This study involves creating Part III of the curriculum, and is mostly based on IBM VisualAge COBOL Version 2.0 with some of the new features in Version 2.2 being referred to as well.

Design of the Visual Builder Tutorial

Organization of the Tutorial Software

The tutorial consists of over 120 screens (pages) and is organized in a book paradigm with three chapters. This form of organization gives the learner a famili feeling that reading the online tutorial is somewhat similar to reading a book, while it puts more learning power in the learner's hands due to the hyper links, Internet access, etc. Also, this organization form takes advantage of the features of the tutorial development tool, ToolBook II Instructor, which is highly suitable for developing book-like (card-and-stack) courseware, as discussed in Chapter Two.

The tutorial starts with a cover page and a contents page. The cover page can be linked to the other three tutorials in the entire VisualAge COBOL curriculum, as described earlier. The contents page links to the first page of each of the three chapters in the tutorial so that the user can quickly locate the beginning of a chapter. Each page in the tutorial has four navigation buttons that link to the next page and the previous page, as well as the first and the last page in the tutorial. Cross-referencing links are used for certain topics, giving the user easy access to referring to other relevant topics, pages, and Internet sites. This makes the tutorial book an open-boundary learning environment.

Several types of objects are used in the pages, besides buttons and links. One is *background*. This is a presentation layout common to several pages. For example, the first page of each chapter has basically the same format, so they can share the same background. Using a common background simplifies the formatting and implementation of pages because objects placed on a background can be shared among pages. Another

type of object is *foreground*. This is the layout unique for each page. Text fields, color graphics, and links are the major objects used in tutoring presentation; and these objects are mostly placed on the foreground.

The User Interface of the Tutorial

Pages are the visual user interface of the tutorial. Figure 3-1 shows a typical page with a color image, text fields, and navigation buttons.

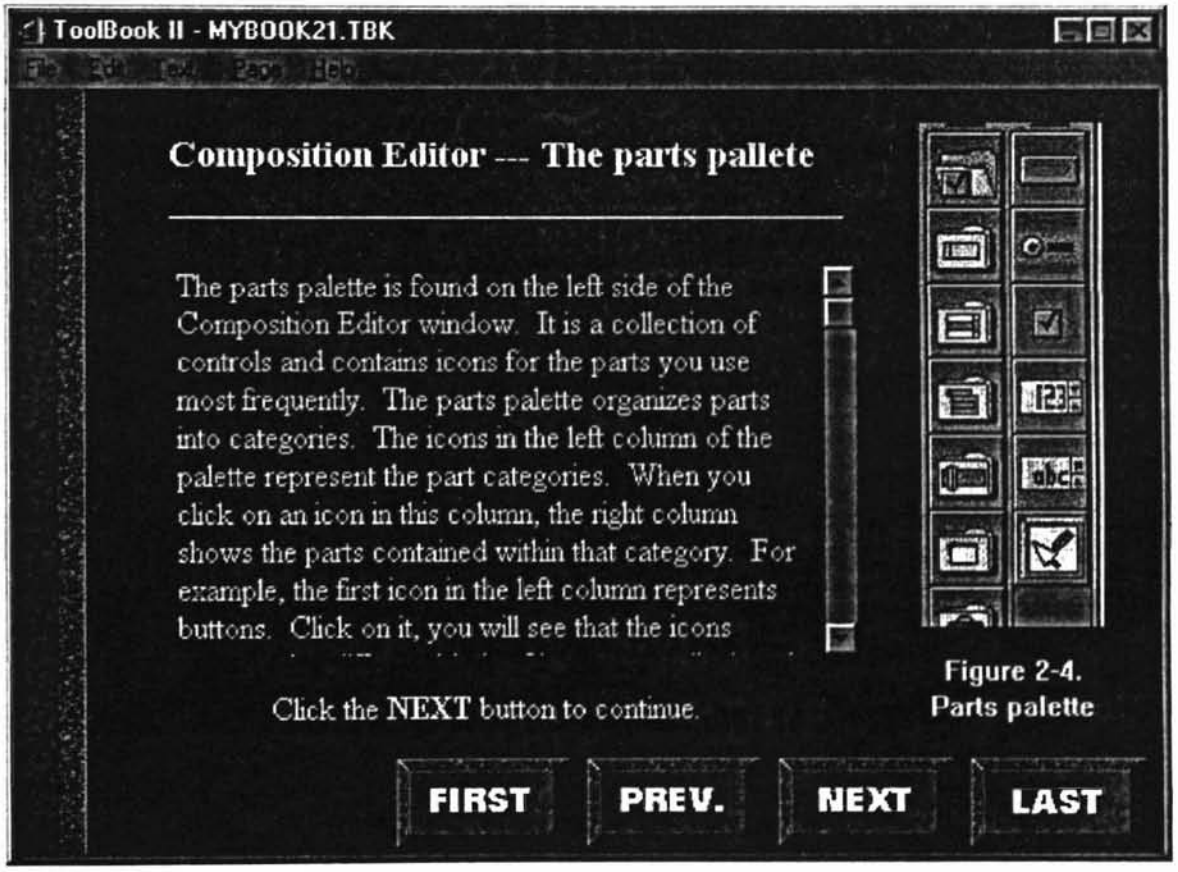

Figure 3-1. A sample page in the tutorial

As shown in Figure 3-1, white text color, green marble margin and buttons, and dark gray background color are used to give the user a relatively clear, relaxed, and comfortable visual effect. Different colors and fonts have been used to indicate hyper links, special terms, or stressed concepts. Vertical scroll bars are enabled in most of the text fields to increase the space utilization and maintain integrity of the text for a particular topic. In most cases, images and graphics related to a topic are managed to appear on the same page where the related text appears, or on the next page. As mentioned before, text fields and other controls are place on the foreground or background of the book. Figure 3-2 shows a page, the same one as shown in Figure 3-1, in its author mode. It illustrates that the text fields on the user interface are selected (surrounded by a frame outlined with squared dots) and ready to be edited.

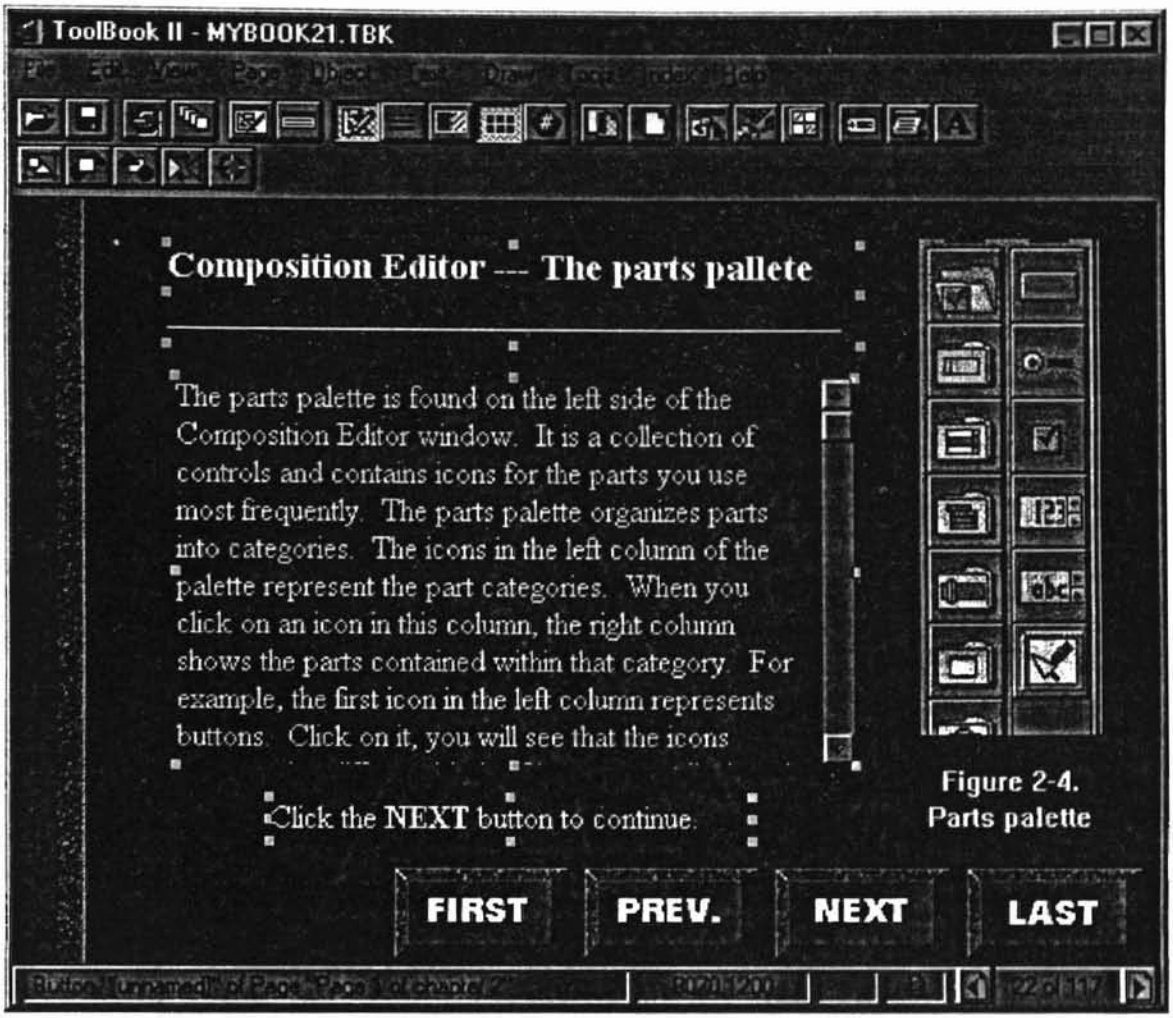

Figure 2-4.
Parts palette

Figure 3-2. A sample page under construction

Contents of the Tutorial

The contents of the tutorial, shown in Table 3-1, were developed according to IBM's references, including "VisualAge for COBOL User's Guide for Windows", "Visual Builder User's Guide", "Getting Started on Windows", and "VisualAge C++ for OS/2" [IBM, 1997b; IBM, 1997d; IBM, 1998b; Carrel-Billiard et al., 1996].

Table 3-1. Contents of the Visual Builder Tutorial

Chapter One: Introduction to Visual Builder
Application Segmentation
What Is Visual Builder?
How to Start Visual Builder When You Create a New Project?
Start Visual Builder from VisualAge COBOL
What Do We Have in the Visual Builder Window?
Loading Part Files
Unloading Part Files
Customizing the Information Area
Selecting/deselecting All Part Files
Import Other Types of Files
Chapter Two: Visual Builder Editors
Introduction
Composition Editor
The tool bar
The parts palette
The free-form surface
Class Editor
Part Interface Editor
The attribute page
The event page
The action page
The promote page
The preferred page

Table 3-1. Contents of the Visual Builder Tutorial (cont'd)

Chapter Three: Developing Applications
Introduction
Object-oriented Approach
Constructing Applications from Parts
What is a part?
Benefits of using parts to construct your applications
Considerations in designing your parts
Working with Parts in Visual Builder Window
Creating a new part
Opening parts
Copying parts from one part file to another
Placing parts on the free-form surface
Placing a part that appears on the parts palette
Adding parts to the parts palette
Using Parts to Build Your GUI
Guidelines for placing parts in the free-form surface
Selecting and deselecting parts
Copying and deleting parts
Displaying pop-up menus
Opening the settings notebook for a part
The pages in the settings notebook
Positioning parts on the grid
Aligning parts and matching part sizes
Listing parts within a composite part
Setting the tabbing order
Promoting a part's features

Table 3-1. Contents of the Visual Builder Tutorial (cont'd)

Sharing Parts with Others
Providing part files (.VCB)
Providing part information files (.VCE)
Connections for Parts
Event-to-attribute connection
Event-to-action connection
Attribute-to-action connection
Browsing Parts' Features When Connecting Them
Browsing a parts features
Determining the source and the target
Making Connections
Connection features to features
Supplying parameter data for incomplete connections
Manipulating Connections
Changing settings for a connection
Reordering connections
Selecting a connection
Changing the source and target of a connection
Deleting connections
Adding Menus to Your Applications
Adding Menus Using Parts Palette
Creating a new Visual Builder project with menus
Building a menu bar with cascade menu items
Connecting menus parts
Building a pop-up menu

Table 3-1. Contents of the Visual Builder Tutorial (cont'd)

Adding Containers
Container parts
Adding a container part
Adding List Boxes
Adding Notebooks
Adding a notebook part
Adding notebook pages
Integrating Visual Parts into a Single Application
Creating dynamic visual parts
Creating static visual parts
Adding visual parts as dynamic instances

Development Environments and Tools

As mentioned before, the development environment for authoring of this tutorial is the Asymetrix ToolBook II Instructor 5.0 [Asymetrix, 1996] software package. It contains various development tools such as Book Specialists, page templates, widgets, script recorder and editor, resource manager, and an array of visual tool palettes. A Book Specialist helps the developer set up the initial structure, layout, and style of the tutorial book, and create a skeleton book. A very powerful tool in ToolBook II Instructor is the collection of widgets. A widget is a ready-made, pre-scripted object that can be dragged and dropped onto a page. Various visual interface objects, such as text fields, graphics,

media clips, and question/answer interactive buttons, can be created using widgets. Script recorder and editor are programming tools used to generate and edit the script for an application. ToolBook II includes a library of OpenScript handlers and code blocks for commonly encountered programming tasks. Also, ToolBook II is capable of converting a courseware developed in the ToolBook II environment into a serious of HTML pages so that they can be used on a Web server immediately. The ToolBook II software used in this study was run on the Windows 95 operating system on a desktop computer.

IBM VisualAge COBOL 2.0 was used to create sample programs and GUI interfaces for demonstration purposes in the tutorial. It was run on the Windows NT Workstation 4.0 operating system on the same desktop computer Large number of color images, created as graphical interfaces in VisualAge COBOL sample projects, were used to illustrate the features and use of Visual Builder. These images were copied and saved as .GIF files using an image editing software, LView. LView is an easy-to-use image editing tool that can run on both Windows NT and Windows 95 platforms.

Implementation of the Tutorial

Skeleton Book

Implementation of the tutorial started with creating a skeleton book with ToolBook II Instructor. The skeleton book was created using the Internet Content Book Specialist in ToolBook II. This book specialist defined the structure and style of the tutorial book with the following major features: a title page, a main contents page with

navigation buttons linked to the first page of each chapter, chapter heading pages, general contests pages with pre-defined navigation buttons, green marble margin and dark gray page background color, and the page size as seven by five inches. Notice that the page size may change after the tutorial book has been converted to HTML pages, depending on the amount of the text on a page and the Web browser being used. This would not cause problems in viewing the pages since scroll bars will be automatically enabled in a Web browser when a page is larger than the size of the browser window.

Initially, the skeleton book was a blank book with three chapters and three blank pages in each chapter. This blank book can be opened in the author mode (i.e. editing mode), activated by selecting menu item Edit => Author in ToolBook II, as shown in Figure 3-3. Based on this, more blank pages were inserted into the book by using Page copy/Paste commands as the development evolved. Various objects, including text fields, graphics, and hyper links, were added onto the blank pages, mostly with a foreground for each page.

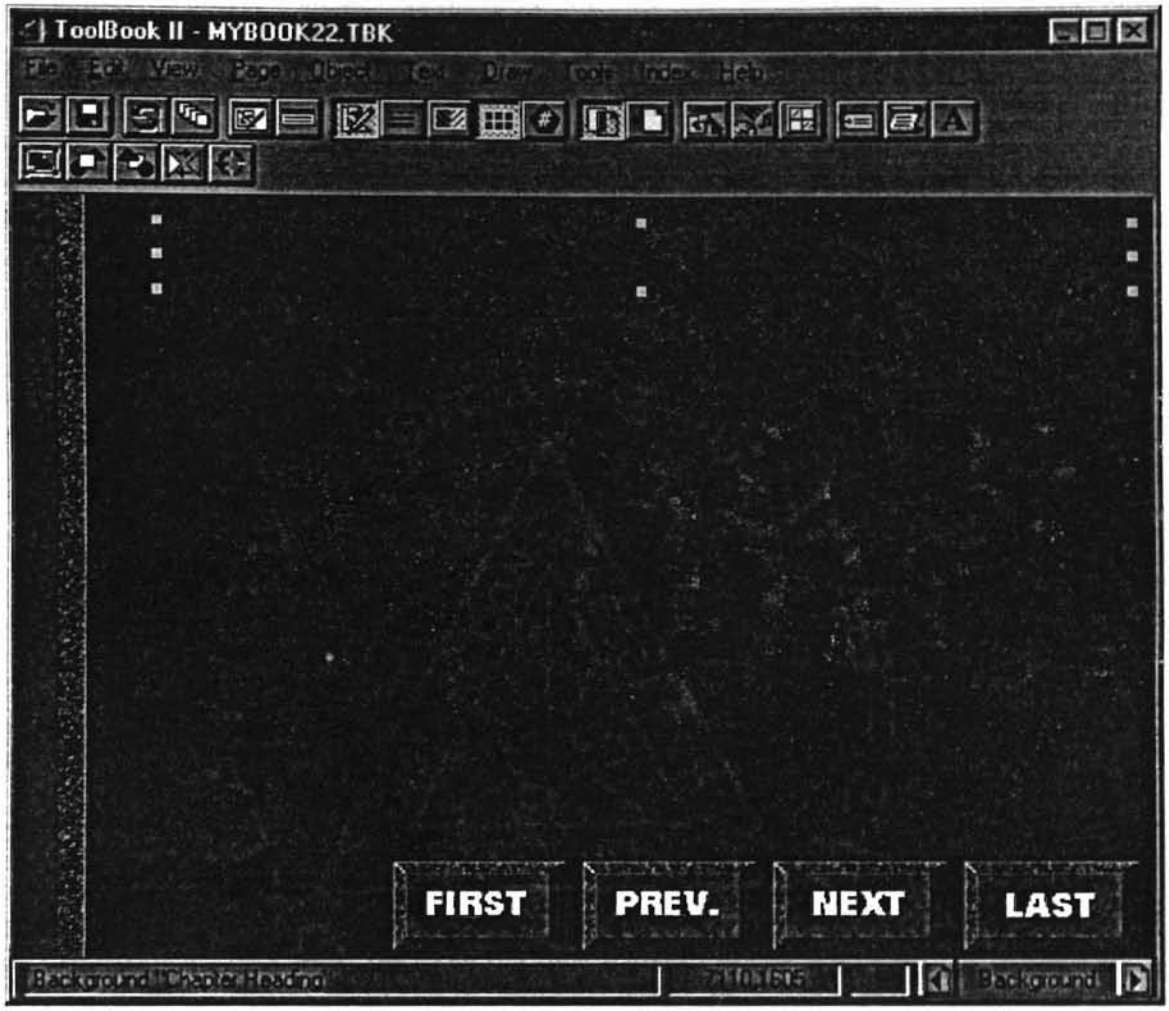

Figure 3-3. A blank page in the author mode

Creating and Editing Objects on Pages

TookBook II provides features that allow the courseware developer to add objects or controls on pages. One of these features can be activated by selecting the menu item Objects => New Widgets, which opens the Internet Widget Catalog window, as shown in Figure 3-4.

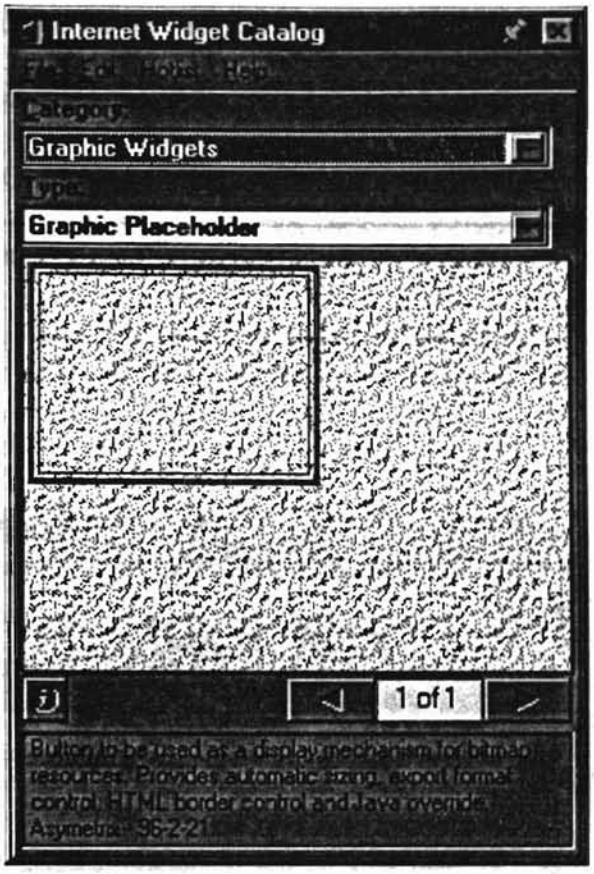

Figure 3-4. The Internet Widget Catalog window

The Internet Widget Catalog window contains various categories and types of controls that are commonly used in Web-base courseware applications. A number of graphic controls in this tutorial were created by selecting the Graphic Widgets category and the Graphic Placeholder type, dragging and dropping the control onto a page, and then importing an image file for the graphic control. In fact, importing an image for a graphic control is a process of defining or modifying the properties of the control. This was done by using the tools in the object property window. In ToolBook II environment, each object is associated with an object property window, which provides access to

editing the properties of the object. Figure 3-5 shows the object property window for a text field. This window was opened by selecting the object first and then selecting the menu item Object => Object Properties => Field Properties. Various editing tasks, such as defining the object name, modifying the border style of the text field, creating or editing event handling script, can be conducted using the tools in this window.

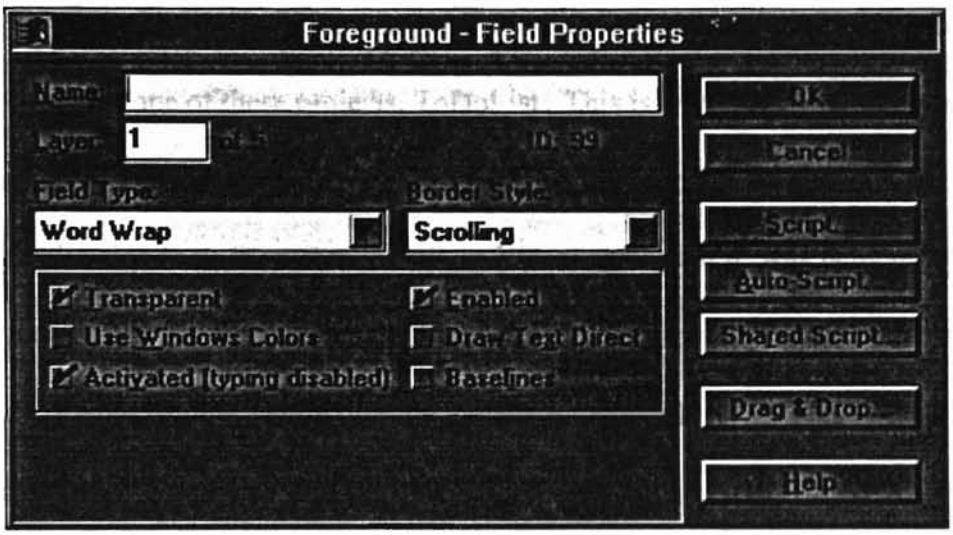

Figure 3-5. Object property window

Another type of controls in ToolBook II is the question/answer interactive widget. This widget implements the functionality that gives the online learner multiple-choice answers to a question. It is a very useful feature in interactive learning. Several objects of this widget type were added on pages at the end of Chapter 3 in the tutorial in order to help the learners review the contents they have learned in this chapter. By clicking on the

multiple-choice answer buttons, the learner gets immediate feedback whether their choice of answer to a question was correct.

Visual Builder Demonstration

In order to illustrate the features, functions, and use of Visual Builder, IBM VisualAge COBOL was run on Windows NT Workstation 4.0 to create sample projects. Several sample projects with GUIs were created using Visual Builder. Figure 3-6 shows the GUI design of one of these projects, ToDoList. This is a typical demo application developed using Visual Builder in VisualAge COBOL. To create this application, a project initialization tool, WorkFrame IDE, in VisualAge COBOL was launched to initialize the project and start the Visual Builder window. Once the new project was established, Visual Builder would provide a blank window object in the Composition Editor, which was the workspace where the GUI parts were created. These parts were obtained from the Parts Palette in the Composition Editor and dropped onto the initially blank window. Editing tools in Visual Builder (e.g. Part Settings Notebook) were used to define the parts' features (attributes, events, and actions) and customize the parts. Connections were made between features of parts, and COBOL source code was generated and edited to make the application viable.

A number of other window images, similar to the one shown in Figure 3-6, were created in the demo applications in a way as described above to illustrate the techniques and procedures in application development using Visual Builder. These images were captured and saved as GIF files using the LView software, and then imported into the

ToolBook II tutorial. That is, these GIF files were used as the resources to create the graphic objects on the pages of the tutorial book so that the learner would be able to see the demo images in the tutorial.

GIF (Graphics Interchange Format) image format was first developed by CompuServ and has the advantage of compressed file size and relatively fast dow nload speed over the Internet [Taylor, 1994]. The images in VisualAge COBOL projects originally captured by LView were in BMP (bitmap) format. Although BMP images tend to have relatively high degrees of resolution, the file size can be very large, which is a significant drawback in downloading files on the Internet. To reduce the image file size and make the Web-based tutorial more viable, the original BMP images were converted to GIF format using LView, resulting in 80 ~ 90% compression in the file size. Figure 3-7 presents a tutorial page containing a graphic object, which shows that the GUI parts shown in Figure 3-6 are under construction.

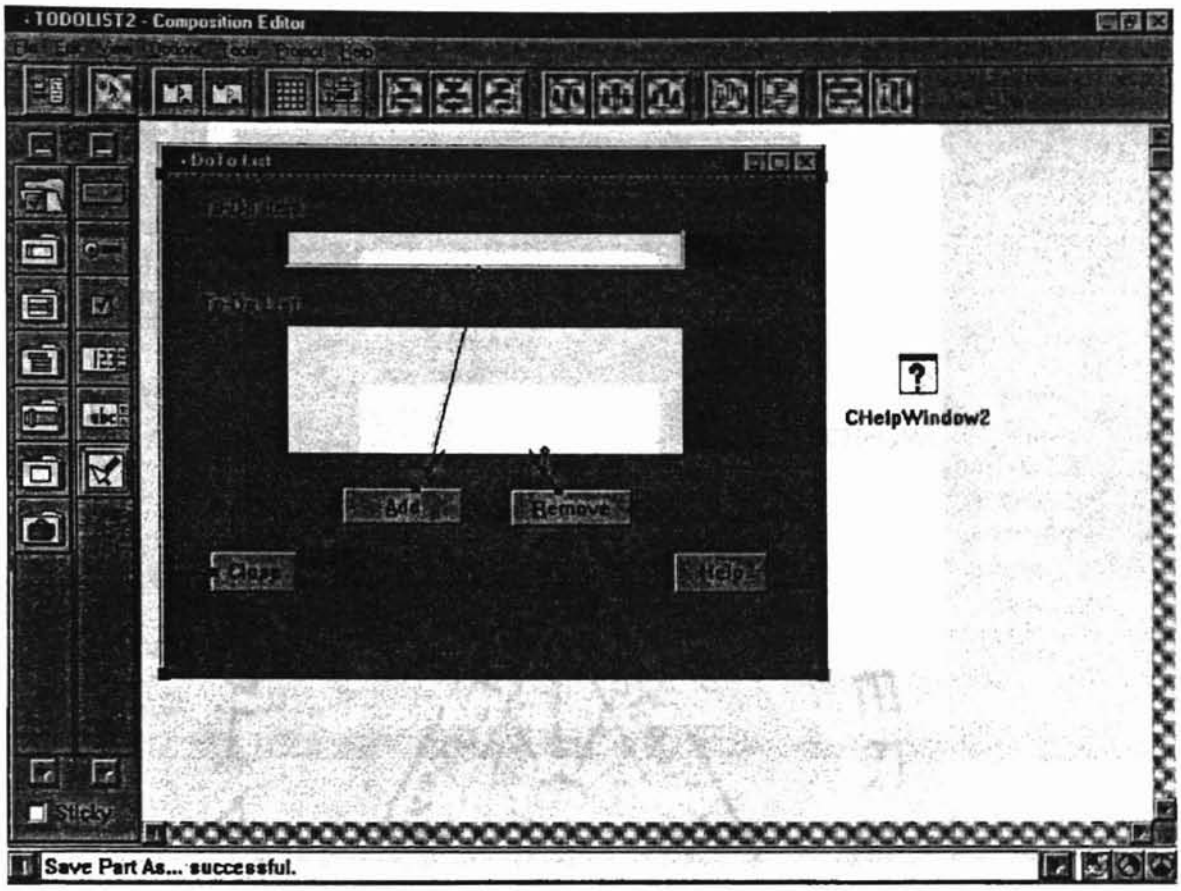

Figure 3-6. GUI development of a demo application using Visual Builder

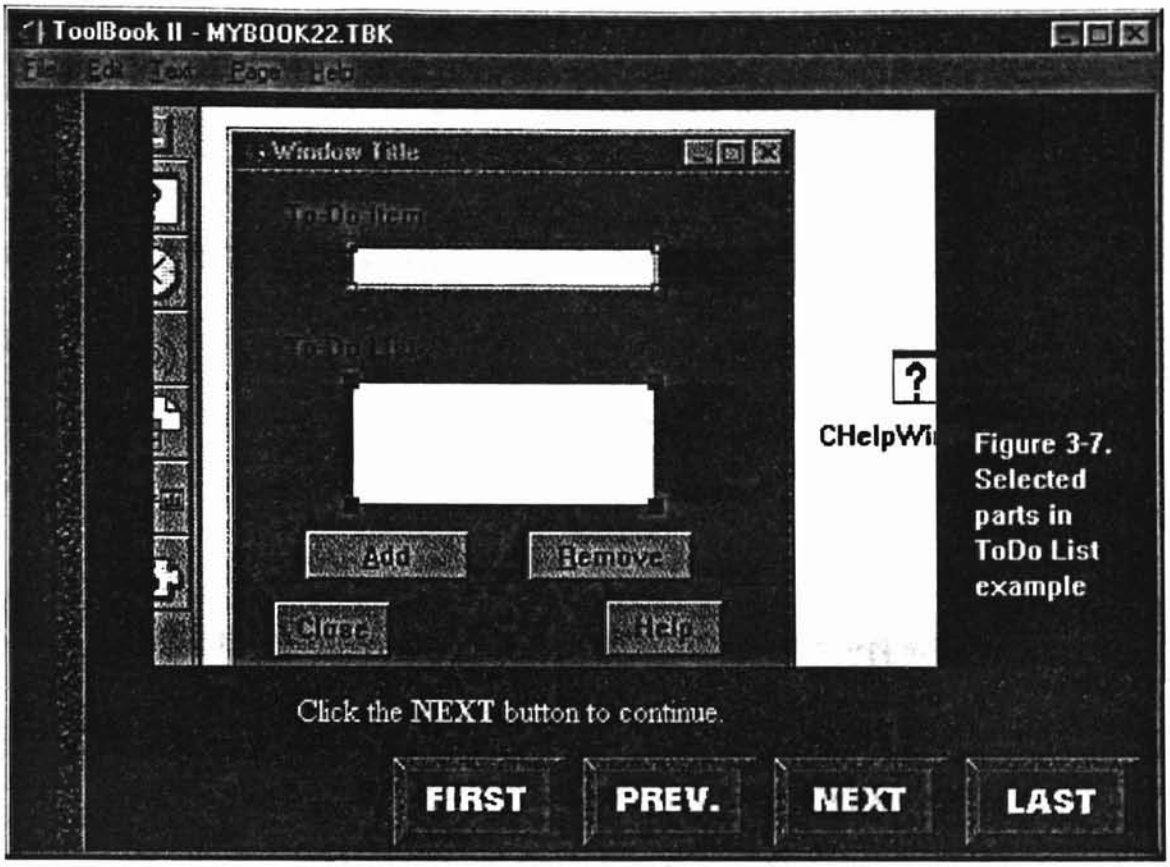

Figure 3-7. A tutorial page showing GUI parts under construction.

Creating HTML Pages from ToolBook II Tutorial Book

Since the ultimate goal of this study was to develop a Web-based tutorial for Visual Builder, the tutorial book (.TBK file) developed in the ToolBook II Instructor environment must be converted to files that could be stored on Web servers and accessed/presented through a Web browser. This has been done by using a tool in ToolBook II which is able to convert a TBK file to a serious HTML files and other accessory files (e.g. image files used in the HTML files). Each of these HTML files

corresponds to a page in the ToolBook II book. As mentioned earlier, the page size, as well as the image size, may change when the ToolBook II book is converted to HTML pages. Therefore, Netscape Composer was used to adjust the layout (such as the size of images and text fields) of the HTML pages for better visual effects. Figure 3-8 shows a page being edited in Netscape Composer. After this, Web browsing programs, including Netscape Navigator 3.0 and 4.0, as well as Microsoft Internet Explorer 3.0, were used to view and test the HTML pages.

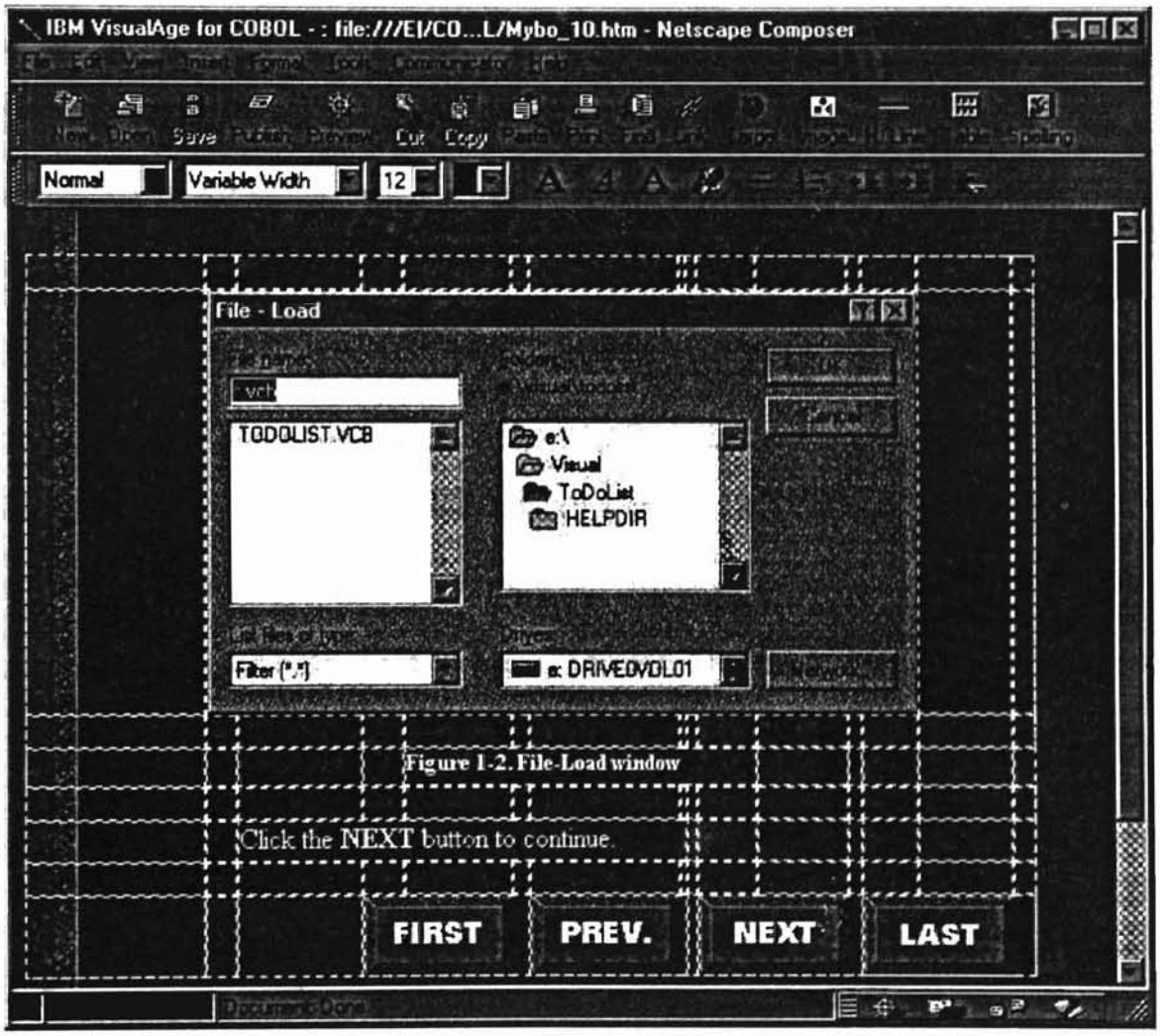

Figure 3-8. A tutorial page being edited in Netscape Composer.

CHAPTER FOUR

RESULTS AND CONCLUSIONS

Results

The final results of this study include the complete Visual Builder tutorial book as a TBK file and the HTML documents (Web pages) derived from the TBK file. The HTML pages and the image files used in the pages require a storage space of approximately 1.2 MB in total. Figures 4-1 and 4-2 show the title page of the tutorial presented by Netscape Navigator 4.0 and Microsoft Internet Explorer 3.0, respectively. Figures 4-3 and 4-4 display another sample page with a graphic object presented in these two Web browsers, respectively. Figure 4-5 shows that a new window of Netscape Navigator was opened on the desktop and connected to a page of the IBM VisualAge COBOL Web site after the link to this site was clicked in the tutorial page (on the right, partially overlapped by the IBM Web site window). The Internet connection software used in this example was America Online 4.0. Figure 4-6 presents a tutorial page at the end of Chapter 3 in the tutorial, which contains a question/answer interactive control with multiple-choice answer buttons. The above results demonstrated that the HTML tutorial pages were functioning as expected.

Figure 4-1. The title page of the tutorial presented in Netscape Navigator 4.0

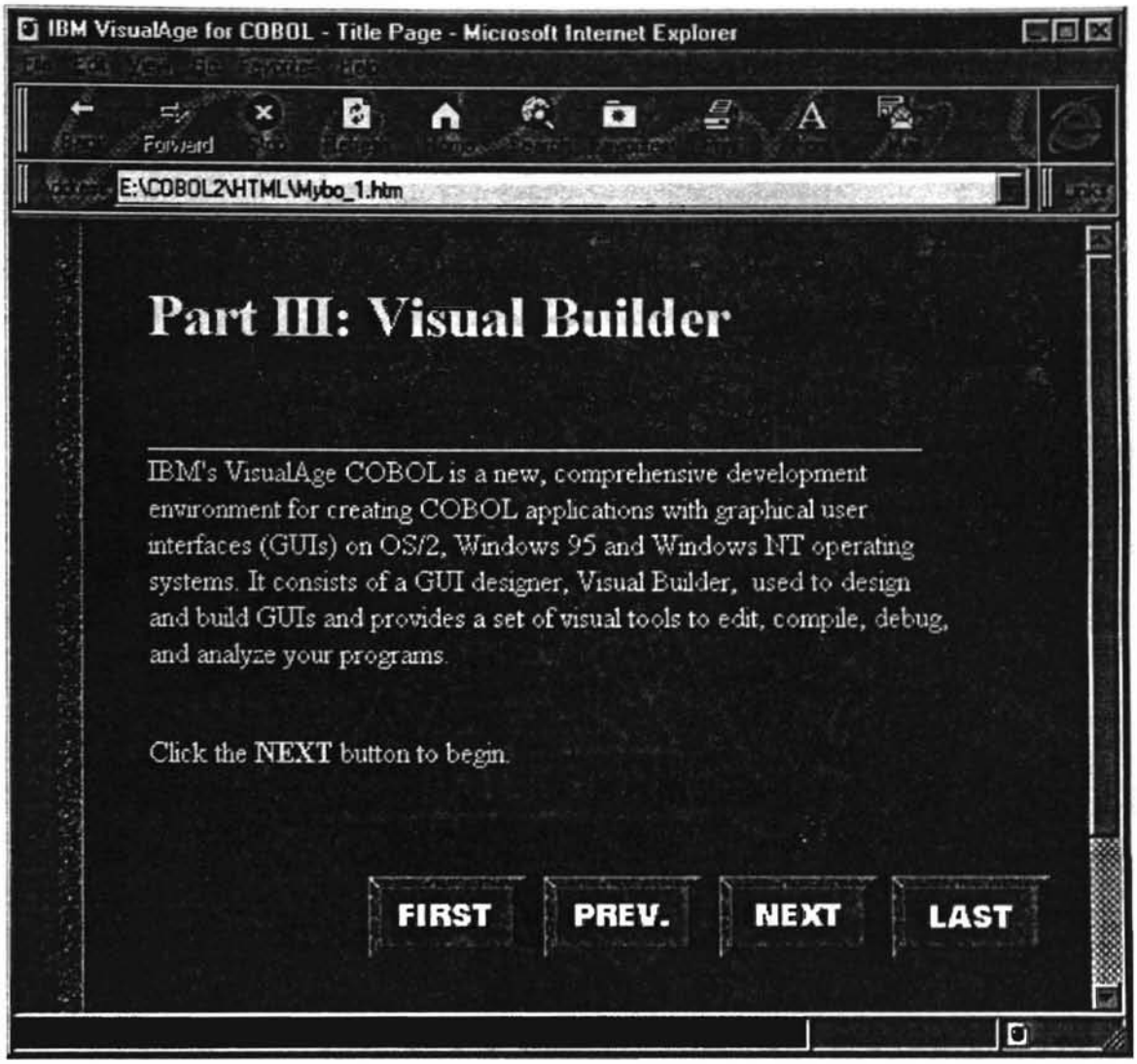

Figure 4-2. The title page of the tutorial presented in Internet Explorer 3.0

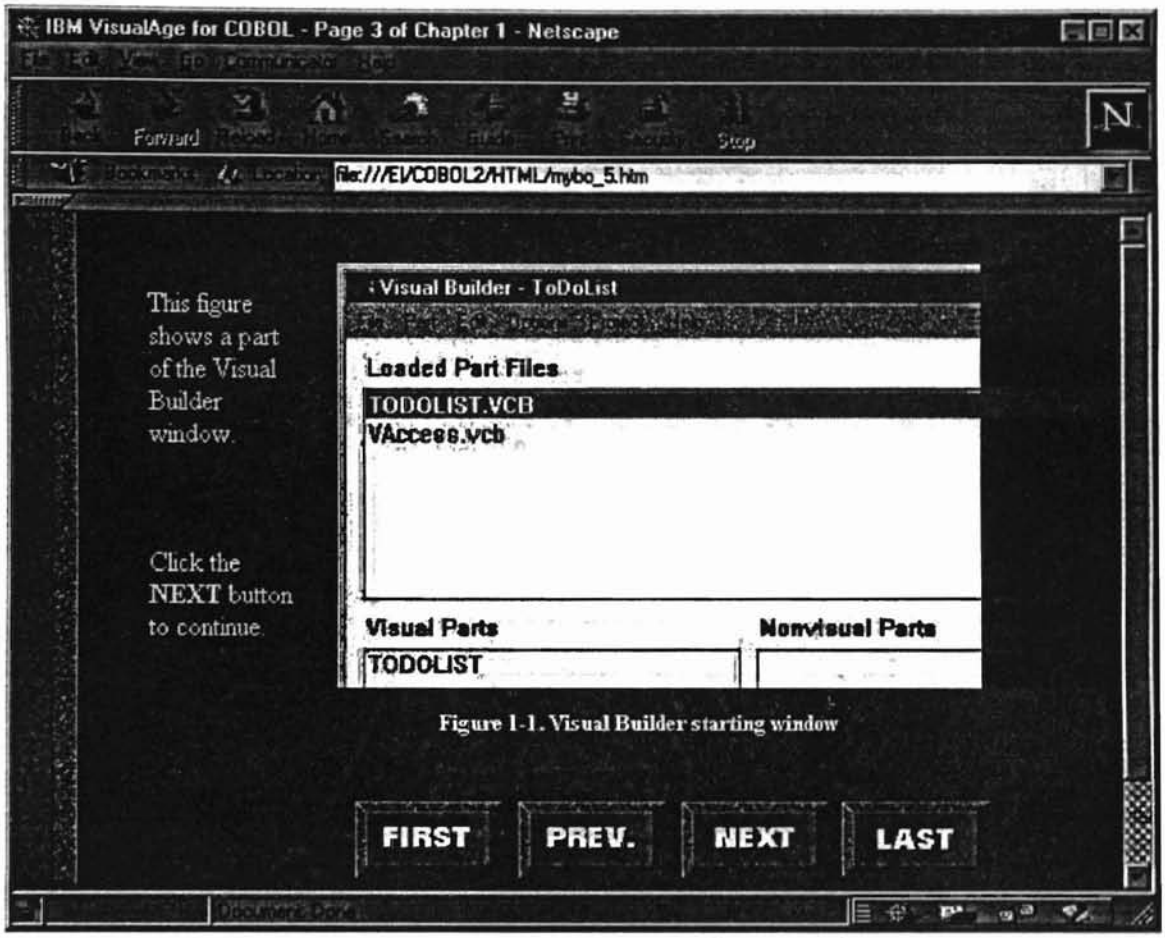

Figure 4-3. A sample page presented in Netscape Navigator 4.0

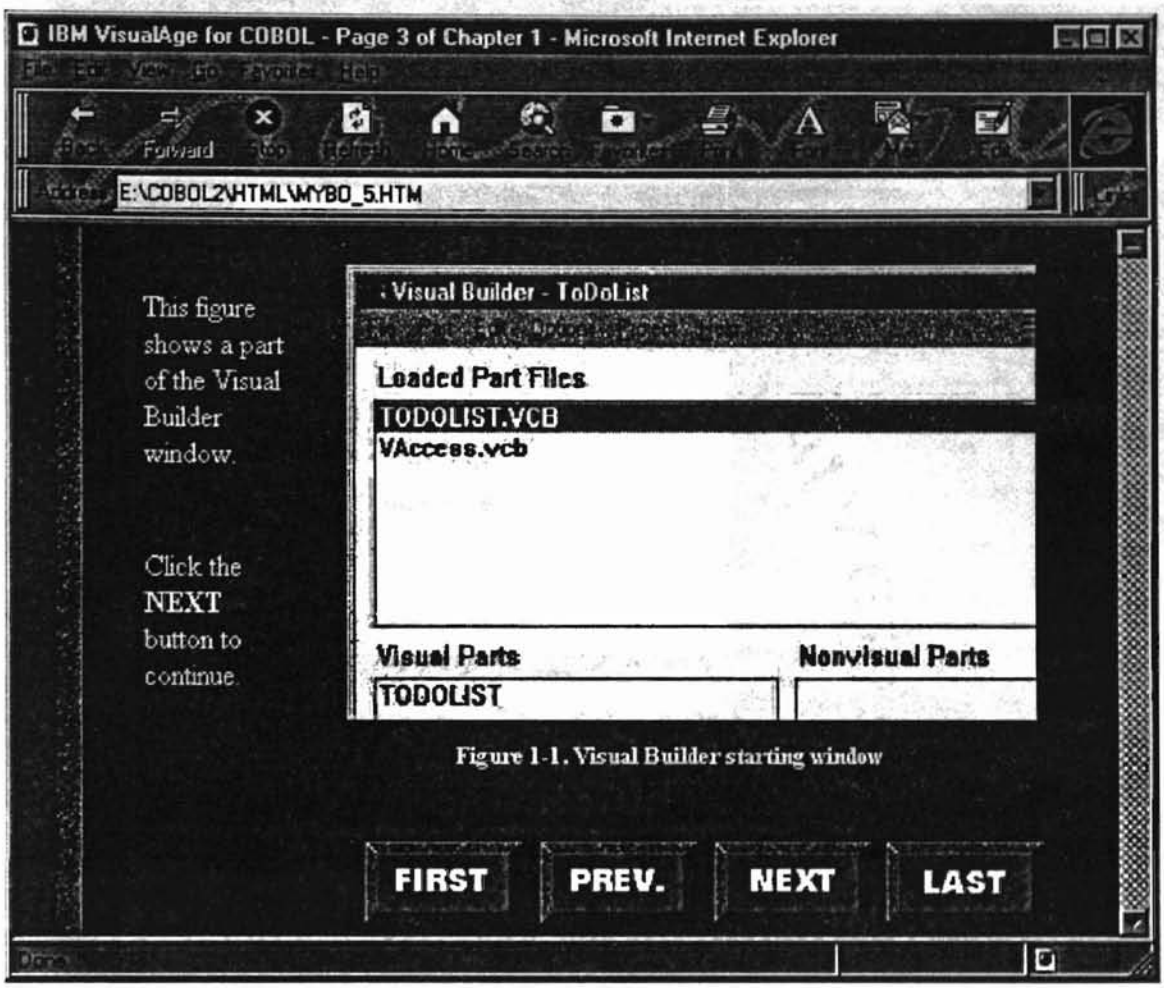

Figure 4-4. A sample page presented in Internet Explorer 3.0

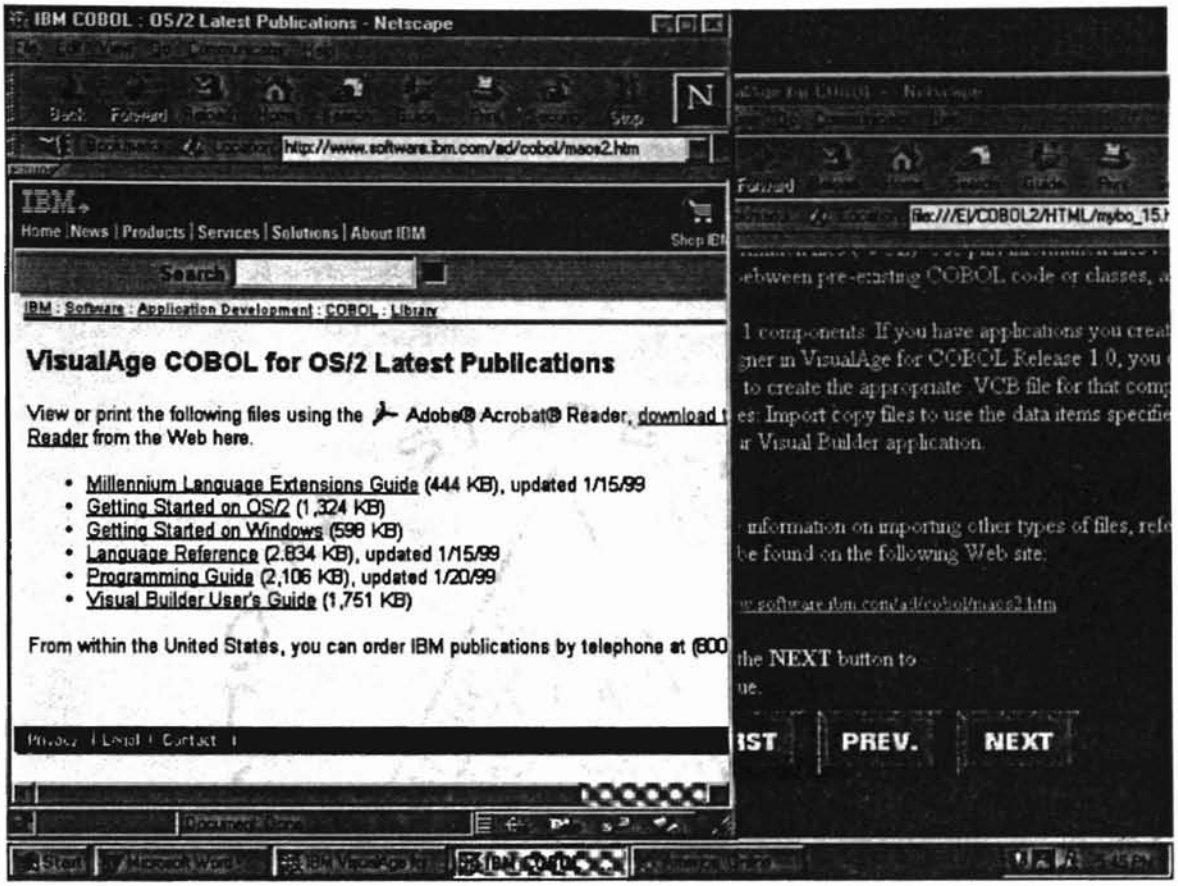

Figure 4-5. An IBM VisualAge COBOL Web page connected from a link in the tutorial

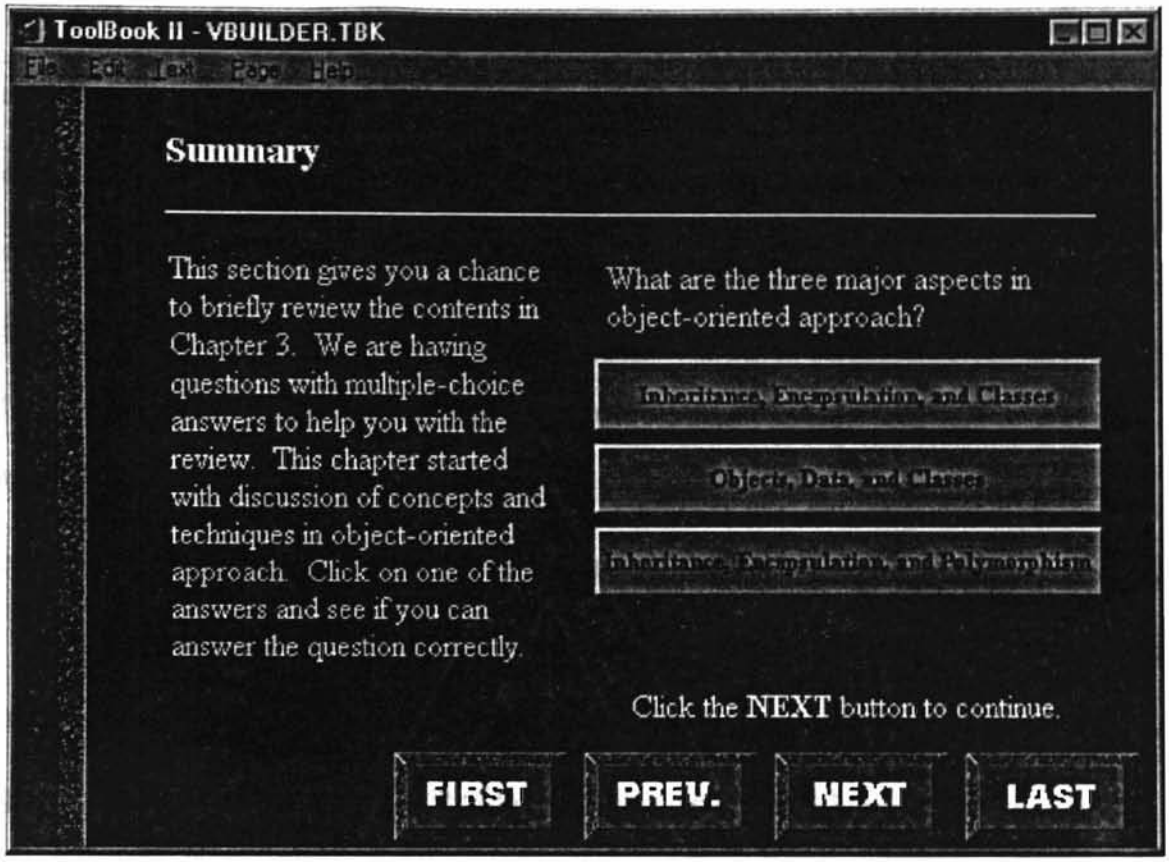

Figure 4-6. A tutorial page containing a question/answer interactive control.

Conclusions

This thesis study has achieved the objective of developing a Web-based tutorial for Visual Builder, the GUI designer in IBM VisualAge COBOL development environment. The tutorial consists of over 120 Web pages (HTML documents) that provide Visual Builder learners with conceptual explanations for GUI design and object-oriented programming approach, step-by-step instructions for using the Visual Builder tools, and description of techniques used in developing Visual Builder applications.

The tutorial was first developed in the ToolBook II Instructor 5.0 development environment as a TBK file, and then converted to HTML files using the HTML file creating tool in ToolBook II. The HTML pages can be satisfactorily presented by commonly used Web browsing software such as Netscape Navigator and Internet Explorer. The entire process of the tutorial development has demonstrated that ToolBook II Instructor is an effective development platform for Web-based courseware applications. Various tools and utilities in ToolBook II make it efficient for the courseware author to create HTML pages used in distance learning and education.

This study has shown that courseware development using ToolBook II allows us to integrate other Web page development tools and techniques into the process of page generation and improvement. In the future enhancement of this tutorial or development of similar online courses, ToolBook II Instructor can be used as a primary development tool to create HTML pages efficiently, especially when a large number of pages with similar page layout need to be created. Based on the ToolBook II -generated pages, it may be reasonable to use other development tools (e.g. Netscape Composer) to modify the page layout and add more HTML tags when necessary. With the help of these tools, it is highly feasible to embed more controls, such as Java applets or ActiveX controls, in the pages to fulfill various requirements of courseware functionality.

REFERENCES

Adams, J., S. Leestma, and L. Nyhoff, 1995. *C++: An Introduction to Computing.* Prentice-Hall, Inc., Englewood Cliffs, NJ.

Asymetrix, 1996. *A Guide to Creating Interactive Courses: Asymetrix ToolBook II Instructor.* Asymetrix Corp., Bellevue, WA.

Bates, A. W., 1995. *Technology, Open Learning and Distance Education.* Routledge, London.

Beekman, G., 1997. *Computer Confluence: Exploring Tomorrow's Technology.* The Benjamin/Cummings Publishing Company, Inc., Menlo Park, CA.

Bouthillier, L., 1998. Synchronized multimedia on the Web. *Web Techniques*, Vol. 3, No. 9, September, pp. 53-55.

Brooks, D. W., 1997. *Web-Teaching: A Guide to Designing Interactive Teaching for the World Wide Web.* Plenum Press, New York.

Budd, T., 1997. *An Introduction to Object-Oriented Programming. Second Edition.* Addison Wesley Longman, Inc., Reading, MA.

Cannon, S. R., 1997. *Understanding Programming --- An Introduction Using C++.* West Publishing Company, St. Paul, MN.

Carrel-Billiard, M., P. Jakab, I. Mauny, and R. Vetter, 1996. *Object-oriented Application Development with VisualAge for C++ for OS/2.* IBM Corporation, San Jose, CA.

Carroll, J. M., J. C. Thomas, and A. Malhotra, 1980. Presentation and representation in design problem-solving. *British Journal of Psychology*, 71:143-153.

Chapin, N., 1997. *Standard Object-Oriented COBOL.* John Wiley and Sons, Inc., New York.

Connick, G. P., 1999. *The Distance Learner's Guide – Western Cooperative for Educational Telecommunications.* Prentice-Hall Inc., Upper Saddle River, NJ.

Cummins, J. and D. Sayers, 1995. *Brave New Schools*. St. Martin's Press, New York, NY.

Ells, J. G. and R. E. Dewar, 1979. Rapid comprehension of verbal and symbolic traffic sign messages. *Human Factors*, 21:161-168.

Galitz, E. O., 1997. *The Essential Guide to User Interface Design: An Introduction to GUI Design Principles and Techniques*. John Wlley & Sons. Inc., New York.

Haykin, R., 1994. *Multimedia demystified: A Guide to the World of Multimedia from Apple Computer, Inc.* Random House, Inc., New York, NY.

Heilman, D., 1999. Distance engineers. *Greater Kansas City's ComputerUser*, Vol. 10, No. 10, February, p. 39.

Herman, D., 1997. Millennial mayhem for manufacturing. *Mechanical Engineering*, Vol. 119, No. 9, pp. 62-65.

Hodges, M. E. and R. M. Sasnett, 1993. *Multimedia Computing: Case Studies from MIT Project Athena*. Addison-Wesley Publishing Company, Inc., Reading, MA.

IBM, 1997a. *VisualAge for COBOL Programming Guide, Version 2.0.* IBM Corporation, San Jose, CA, SC26-9050-00.

IBM, 1997b. *VisualAge for COBOL Getting Started on Windows, Version 2.0.* IBM Corporation, San Jose, CA, GC26-8944-00.

IBM, 1997c. *VisualAge for COBOL Building Parts for Fun and Profit, Version 2.0.* IBM Corporation, San Jose, CA, GC26-9038-00.

IBM, 1997d. *VisualAge for COBOL User's Guide for Windows, Version 2.0.* IBM Corporation, San Jose, CA, SC26-9037-00

IBM, 1998a. IBM application development. http://www.software.ibm.com/ad/.

IBM, 1998b. *Visual Builder User's Guide (Third Edition)*. IBM Corporation, San Jose, CA, GC26-9053-02.

IBM-ITSO, 1996. *IBM VisualAge for COBOL for OS/2 Objected-Oriented Programming*. SG24-46-6-00.

Jezequel, J-M., 1996. *Object-Oriented Software Engineering with Eiffel*. Addison-Wesley Publishing Company, Inc., Reading, MA.

Kay, A. C., 1993. The early history of smalltalk. The Second ACM SIGPLAN History of Programming Languages Conference (HOPL-II), *ACM SIGPLAN Notices* 28(3):69-75, March, 1993.

Levey, R., 1996. *Reengineering COBOL with Objects*. McGraw-Hill, New York, NY.

Longhurst, J. and A. Longhurst, 1989. *COBOL*. Prentice Hall, Englewood Cliffs, NJ.

Moor, M. G. and M. M. Thompson, 1990. The effects of distance learning: a summary of the literature. *Research Monographs*, No. 2, American Center for the Study of Distance Education, Pennsylvania State University, University Park, PA.

Musciano, C. and B. Kennedy, 1997. *HTML: The Definitive Guide*. O'Reilly and Associates, Inc., Cambridge, UK.

Myers, B. A. and M. B. Rosson, 1992. Survey on user interface programming. *Proceedings: CHI 92 (Conference on Human Factors in Computing Systems)*. May 3-7, Monterey, CA.

Nickerson, R. C., 1987. *Fundamentals of Structured COBOL, Second Edition*. Scott, Foresman and Company, Glenview, IL.

Porter, L. R., 1997. *Creating the Virtual Classroom: Distance Learning with the Internet*. John Wiley & Sons, Inc., New York.

Rogerson, D., 1997. *Inside COM - Microsoft's Component Object Model*, p19. Microsoft Press, Redmond, WA.

Rumbauch, J., M. Blaha, W. Premerlani, F. Eddy, and W. Lorensen, 1991. *Object-oriented Modeling and Design*, p. 4. Prentice-Hall, Inc., Englewood Cliffs, NJ.

Taylor, R. B., 1994. *The Multimedia Home Companion: 400 Ratings and Reviews*. Warner Books, Inc., New York, NY.

Tullis, T., 1983. Predicting the usability of alphanumeric displays. Ph.D. dissertation, Rice University.

Van den Brande, L., 1993. *Flexible and Distance Learning*. John Wiley & Sons, Chichester, UK.

Wang, M., 1998. *Creating a Web-Based Tutorial for IBM VisualAge for COBOL*. Master's thesis, Oklahoma State University.

Williams, A., 1998. Coming of visual age. *Web Techniques*, Vol. 3, No. 10, October, p. 36.

VITA

Jiazheng Li

Candidate for the Degree of

Master of Science

Thesis: DEVELOPMENT OF A WEB-BASE TUTORIAL FOR VISUAL BUILDER --- THE GUI DESIGNER IN IBM VISUALAGE COBOL

Major Field: Computer Science

Biographical:

Education: Graduated from Tsinghua University, Beijing, China, with a Bachelor of Engineering degree in Environmental Engineering in June, 1984; received a Master of Science degree in Environmental Science at Nanjing University, Nanjing, China, in July, 1987; received a Doctor of Philosophy degree in Civil and Environmental Engineering at Oklahoma State University, Stillwater, Oklahoma, in July, 1997; completed the requirements for the Master of Science degree in Computer Science at Oklahoma State University in May, 1999.

Experience: Raised in the city of Changsha, Hunan Province, China; employed as a lecturer in Environmental Engineering at Hunan University, Changsha, China, 07/87 to 12/ 92; worked as a research and teaching assistant in Computer Science as well as in Environmental Engineering at Oklahoma State University, 08/93 to 5/98; employed as an information systems application developer at Cerner Corporation, Kansas City, Missouri, 08/98 to present.

Professional Memberships: American Society of Civil Engineers; American Chemical Society.